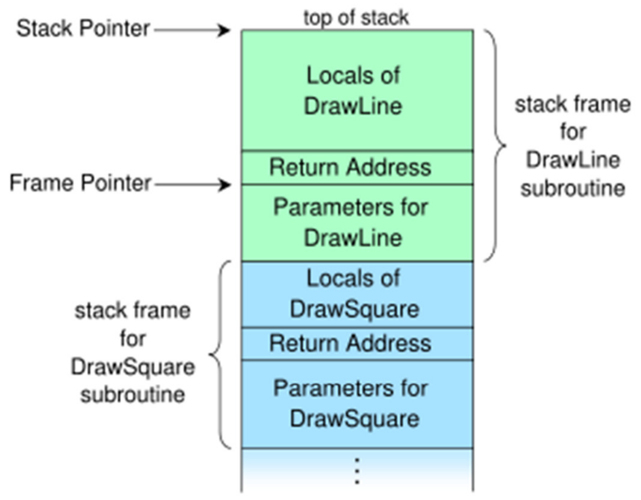
Copied from https://en.wikipedia.org/wiki/Call_stack#Structure

---

A call stack is composed of **stack frames** (sometimes called **activation records**). These are machine dependent data structures containing subroutine state information. Each stack frame corresponds to a call to a subroutine which has not yet terminated with a return. For example, if a subroutine named `DrawLine` is currently running, having just been called by a subroutine `DrawSquare`, the top part of the call stack might be laid out like this (where the stack is growing towards the top):



The stack frame at the top of the stack is for the currently executing routine. In the most common approach the stack frame includes:

- space for the local variables of the routine,
- the return address back to the routine's caller, and
- the parameter values passed into the routine.

The primary purpose of a call stack is:

- **Storing the return address** – When a subroutine is called, the location of the instruction to return to needs to be saved somewhere. Using a stack to save the return address has important advantages over alternatives. One is that each task has its own stack, and thus the subroutine can be reentrant, that is, can be active simultaneously for different tasks doing different things. Another benefit is that recursion is automatically supported. When a function calls itself recursively, a return address needs to be stored for each activation of the function so that it can later be used to return from the function activation. This capability is automatic with a stack.

A call stack may serve additional functions, depending on the language, operating system, and machine environment. Among them can be:

- **Local data storage** – A subroutine frequently needs memory space for storing the values of local variables, the variables that are known only within the active subroutine and do not retain values after it returns. It is often convenient to allocate space for this use by simply moving the top of the stack by enough to provide the space. This is very fast compared to heap allocation. Note that each separate activation of a subroutine gets its own separate space in the stack for locals.
- **Parameter passing** – Subroutines often require that values for parameters be supplied to them by the code which calls them, and it is not uncommon that space for these parameters may be laid out in the call stack. Generally if there are only a few small parameters, processor registers will be used to pass the values, but if there are more parameters than can be handled this way, memory space will be needed. The call stack works well as a place for these parameters, especially since each call to a subroutine, which will have differing values for parameters, will be given separate space on the call stack for those values.

- **Enclosing subroutine context** - Some programming languages (e.g., Pascal and Ada) support nested subroutines, allowing an inner routine to access the context of its outer enclosing routine, i.e., the parameters and local variables within the scope of the outer routine. Such static nesting can repeat - a function declared within a function declared within a function... The implementation must provide a means by which a called function at any given static nesting level can reference the enclosing frame at each enclosing nesting level. Commonly this reference is implemented by a pointer to the encompassing frame, called a "downstack link" or "static link", to distinguish it from the "dynamic link" that refers to the immediate caller (which need not be the static parent function). For example, languages often allow inner routines to call themselves recursively, resulting in multiple call frames for the inner routine's invocations, all of whose static links point to the same outer routine context. Instead of a static link, the references to the enclosing static frames may be collected into an array of pointers known as a *display* which is indexed to locate a desired frame. The Burroughs B6500 had such a display in hardware that supported up to 32 levels of static nesting.
- **Other return state** – Besides the return address, in some environments there may be other machine or software states that need to be restored when a subroutine returns. This might include things like privilege level, exception handling information, arithmetic modes, and so on. If needed, this may be stored in the call stack just as the return address is.